Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 278 (2011) 187–199

www.elsevier.com/locate/entcs

# LotrecScheme

François Schwarzentruber[2]

*IRISA / ENS Cachan (Bretagne), France*

**Abstract**

We propose a generic tableau prover extending Lotrec, called LotrecScheme: the tool enables to enrich tableau method rules with executable code. The user can graphically design its own tableau method and can solve the problem of model construction. This new system provides a way to merge two worlds when the system check a loop (for instance for S4): merging instead of blocking has the benefit of providing a real model and not just a skeleton of the model. The system generalizes also the notion of satisfiability to labeled graphs.

*Keywords:* Modal logic. Tableau method. System description.

## 1 Introduction

One major decision problem in logic is the satisfiability problem: given a formula $\varphi$, we want to know whether there exists a model satisfying the formula $\varphi$. Tableau methods [4] often give efficient procedures for this problem. That is why researchers may need tools to test their tableau methods. That is why teachers need tools for demonstrating to students how tableau methods work. Here we propose a tool to write tableau methods that fulfills the following requirements:

- *generic tool.* It provides a framework to design many customized logics: it captures many syntaxes [3] of formulas in logic and provides data structures needed to implement the corresponding tableau method. The tool allows to add some executable code to the tableau rules in order to deal with integers, lists, sets, etc.
- *Easy implementation.* Researchers/teachers/students may not be familiar with a programming language like C/Java/OCaml/Scheme etc. So the tool must provide a high-level pattern-matching mechanism and a graphical user interface in order

[1] Thanks to Olivier Gasquet, Andreas Herzig and Bilal Saïd for their useful discussions.

[2] Email: francois.schwarzentruber@bretagne.ens-cachan.fr

[3] $K_a\varphi$ in epistemic modal logic, $\varphi U \psi$ in temporal logics, $[(a^*; b)^*]\varphi$ in Propositional Dynamic Logic etc.



doi:10.1016/j.entcs.2011.10.015

to design a tableau method. No programming skill is needed to design a method for simple modal logics as K.

- *debugging.* Of course, as we need to test/understand/teach a tableau method, we need a tool that displays step-by-step the construction of the model during the tableau method. In addition to checking whether $\varphi$ is satisfiable, you can also check whether a given labeled graph is satisfiable, that is to say whether we can extend it into a model.

We fill the gap by providing a user interface where the user can draw tableau rules graphically, a step-by-step debugger and an engine where tableau rules can be enriched with executable code. The language we chose to implement the generic prover is the LISP dialect Scheme [5] for many reasons. Generally speaking tableau methods deal with terms and pattern-matching thus symbolic expressions (s-expressions) of Scheme language are suitable for this issue. It also enables mixing data such as formulas and executable code with quote and unquote operations [5] and thus enables the user to customize a tableau method. The language is dynamically typed hence it is adapted for prototyping a tableau method. The user graphical interface is written in JAVA for multi-platform reasons. The engine in Scheme is called by JAVA via Kawa [3]. The software is available at the following URL:

http://www.irisa.fr/prive/fschwarz/lotrecscheme/.

## 2 Related works

Molle [4] and OOPS [17] are *specific* provers for respectively modal logic K(T) and $S5_n$ but have nice graphical interface views to interact with the prover.

The Logic Workbench [5] [11] is a powerful formal tool to manipulate logics. Many libraries are provided for K, S4, S5, etc. Nevertheless, it requires advanced programming skills to implement a new method for a new logic.

Tableau WorkBench TWB [6] [1] is a powerful generic tool written in OCaML. It requires to have programming skills in OCaml. It confirms that functional languages are suitable for tableau method.

**Comparison with LoTREC**

LoTREC [7] ([6], [8]) is a generic prover written in Java for modal logics. Some results of terminaison and completeness of tableau methods written in the LoTREC language has been proven [9]. It captures an impressive variety of modal logics [16]. LoTREC is a tool that relies on graph rewriting rules. Such a rule replaces a given graph pattern in a whole graph with another one. LotrecScheme is directly inspired by LoTREC is the

[4] http://molle.sourceforge.net/
[5] http://www.lwb.unibe.ch/
[6] http://twb.rsise.anu.edu.au/
[7] http://www.irit.fr/Lotrec/

sense that it works as it is described in [10]. LoTREC has been designed for educational purposes [7]. As LoTREC, LotrecScheme also provides a pattern-matching mechanism and allows to define loop checking mechanisms.

Nevertheless, there are some limitations in the pattern-matching mechanism of LoTREC. First, it is impossible to identify a disconnected pattern. For instance, it is impossible to write a rule where the condition is 'if the current tableau contains a world $w$ and a world $u$...'. In LoTREC $w$ and $u$ would have to be linked by a relation [8]. Secondly, it is impossible to have pattern-matching on the name of worlds [9]. LotrecScheme has a general pattern-matching mechanism that fixes those issues (see Section 3 for an example with pattern-matching over worlds' names).

Formulas in LoTREC are written in prefix style whereas LotrecScheme can use prefix, infix, postfix notations as long as expressions are fully parenthesized. For instance, the formula of Propositional Dynamic Logic $[a*;b](p \wedge q)$ may be represented by '`box seq star a b and p q`' in LoTREC and by the s-expression '`(box ((a *) ; b) (p and q))`' in LotrecScheme.

As LoTREC, LotrecScheme provides a graphical interface to display the tableau and the execution of the tableau method. But LotrecScheme also provides a user graphical interface so that the user sees and designs tableau rules dealing with graphs. The role of the graphical interface is essentially pedagogical.

Concerning the engine itself, LoTREC it is impossible to evaluate dynamically expressions written by the user while integers, lists, sets and other extra data structures are often used in tableau methods (see Section 4). As far as I know, dynamic evaluation is difficult to implement in Java. Contrary to LoTREC, LotrecScheme allows to enrich tableau rules with executable code written in Scheme. Technically this feature is done with the `eval` function. So LotrecScheme fills the gap and is a generic prover in which it is easy to deal with integers, lists, sets, hash-tables etc.

LoTREC solves the model construction problem for a given formula $\varphi$, that is, we want to construct a model containing a node in which $\varphi$ is satisfied. Contrary to LoTREC, LotrecScheme also solves the more general model construction problem of a given graph, that is, to construct a model extending a given graph such that every formula $\varphi$ in a node $w$ is satisfied in $w$.

Unfortunately contrary to LoTREC, it is not possible yet to write arbitrary strategies in the current version of LotrecScheme. The only strategy implemented in LotrecScheme consists in applying the first applicable rule of a given list of rules as long as possible and stop when there is no applicable rule anymore.

But, this strategy is sufficient and the current version of LotrecScheme enables to already implement many tableau methods for various modal logics: $K$, $KT$, $KD$, $S4$, $S4.3$, $S5_n$ ([12]), $[4]S5_n$-PAL ([2]), $KD45_n \oplus \Box_i\varphi \rightarrow \Box_{i+1}\varphi$... To sum up, LotrecScheme combines the graphical aspect of Molle, OOPS, the philosophy of writing graph rewriting rules of Lotrec and a rich programming language like LWB

[8] It is always possible to access 'disconnected' worlds by constructing the universal relation, but this makes the tableau method artificial.

[9] It is always possible to simulate the name of the worlds by adding a specific kind of formulas in the node as 'name ...' but this makes the tableau method artificial.

and TWB.

## 3 Genericity: writing rules in LotrecScheme

A tableau method in LotrecScheme is a rewriting system of a set of terms (LoTREC was already modeled in such a way in [10]). A rewriting rule is made up of a left-hand side (the pattern we have to find in the model in construction) and a right-hand side (how the pattern is replaced). Terms are here represented by Scheme expressions called s-expressions defined by induction as follows:

- Numbers and identifiers are s-expressions;
- If $s_1, \ldots, s_n$ are s-expressions then $(s_1\ s_2\ \ldots s_n)$ is an s-expression.

Formulas, agents, programs of Propositional Dynamic Logic, etc. are also represented by s-expressions. Upper case identifiers are variables that are instanciated during the pattern-matching process. The rules can be:

- deterministic as the $\Diamond$-rule:

$$\frac{(A, \Diamond\varphi)}{(A, link, B)(B, \varphi)}$$

  where $A$ and $B$ represent world variables and $\varphi$ represents a formula variable. This rule is represented by the following s-expression:

```
(primitiverule
  (condition ((formula A (diamond PHI))))
  (add ((formula B PHI) (A link B) (world B))))
```

- non-deterministic, as the Or rule

$$\frac{(A, \varphi \vee \psi)}{(A, \varphi)|(A, \psi)}$$

  represented by the following s-expression:

```
(primitiverule (condition ((formula A (PHI1 or PHI2))))
               (add-nondeterminist ((formula A PHI1))
                                   ((formula A PHI2))))
```

- or an halting rule, as the Bottom rule

$$\frac{(A, \bot)}{halt}$$

  represented by the following s-expression:

```
(primitiverule (condition ((formula A bottom)))
               (add ((halt)) ) )
```

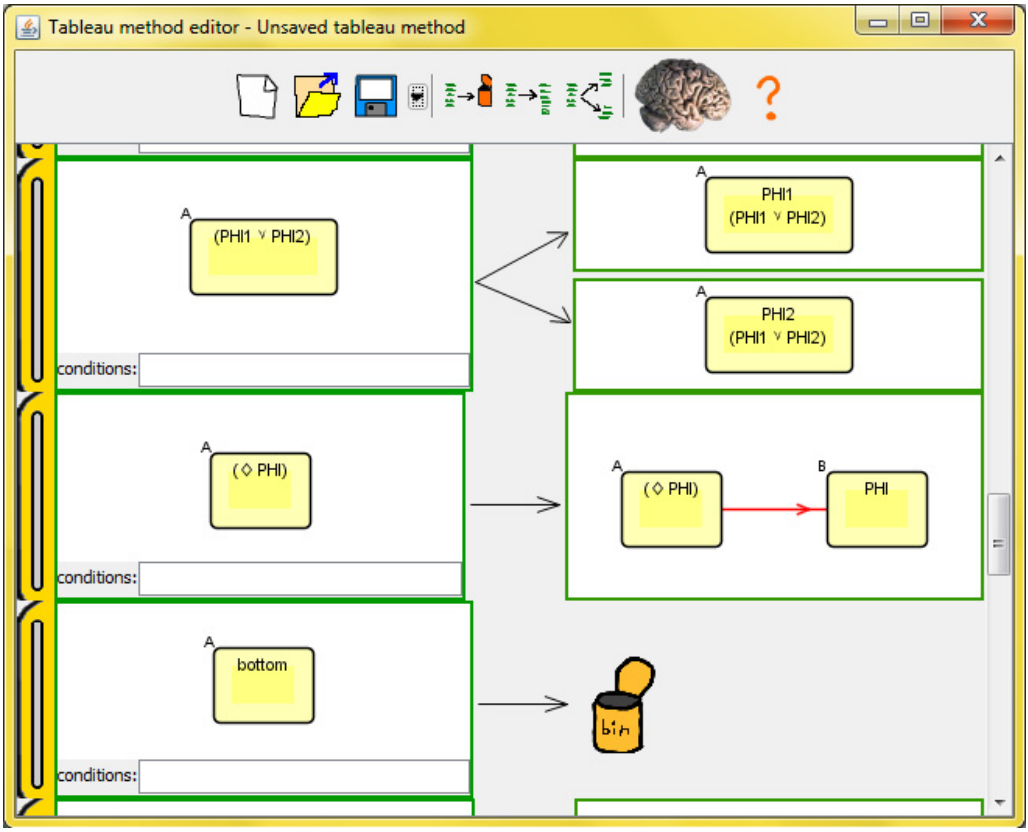



Fig. 1. The window for editing the tableau method

The rewriting rules related to graph rewriting are designed by the user graphically: adding nodes, edges, formulas, terms to the left-hand side or right-hand side. The Figure 1 depicts the graphical tableau method editor and shows here the ◇-rule, the Or-rule and the Bottom rule. This window enables the user to add nodes, terms, edges and rename objects or to delete objects by mouse operations. In the background, the graphical editor generates terms of the following form:

- (world w): $w$ is a world;
- (formula w $\varphi$): the world $w$ contains the formula $\varphi$;
- (w link u): the world $w$ is linked to the world $u$ by $R$;
- (w link a u): the world $w$ is linked to the world $u$ by $R_a$.

Other terms are not displayed by the graphical user interface. The user may imagine other kind of terms like (agent a) ($a$ is an agent), (rel w u v a) (the triple $(w, u, v)$ belongs to the ternary relation $R_a$) etc. All kinds of terms are supported by the engine of LotrecScheme but not by the graphic user interface yet: the user has to edit her tableau method file in a text editor.

The order of the rules determines a canonical strategy of rules applications: LotrecScheme loops by applying the applicable first rule of the list until there is

no applicable rule anymore. More precisely, given a sequence of rules $r_1, \ldots, r_n$, LotrecScheme applies the following strategy:

**while** there exists a rule $r_i$ is applicable
  **if** $r_1$ is applicable
    apply $r_1$
  **else if** $r_2$ is applicable
    apply $r_2$
  ⋮
  **else if** $r_n$ is applicable
    apply $r_n$
  **endIf**
**endWhile**

It would be interesting to know when this strategy is not sufficient.

**Pattern-matching over worlds' names**

The reader may find a tableau method for reasoning about Dynamic Epistemic Logic sequents $(\varphi, \varphi', \varphi'')$ on the webpage

http://www.irisa.fr/prive/fschwarz/publications/m4m2011/.

Such a triple $(\varphi, \varphi', \varphi'')$ is satisfiable iff there exists pointed models $\mathcal{M}, w$ and $\mathcal{M}', w'$ such that $\mathcal{M}, w \models \varphi$, $\mathcal{M}', w' \models \varphi'$ and $\mathcal{M} \otimes \mathcal{M}', (w, w') \models \varphi''$ where $\mathcal{M} \otimes \mathcal{M}', (w, w')$ 'kind of' cartesian product of $\mathcal{M}$ and $\mathcal{M}'$.

One of the tableau rule concerns the $\Diamond$-rule in a world $(a, b)$ of the cartesian product that consists in adding successors to $a$ and $b$:

```
(primitiverule
  (condition ((formula (A B) (diamond PHI))))
  (add ((formula (C D) PHI) (formula (C D) ok)
        (A link C) ((A B) link (C D)) (B link D)
        (world (C D)) (world A) (world C) (world B)
        (world D))))
```

Contrary to LotrecScheme, in LoTREC it is impossible to do pattern-matching on the name of the world that is a pair $(a, b)$.

LotrecScheme enables to apply rule step-by-step as the Figure 2. It gives the next rule to apply (here the $\Diamond$-rule) to the model in construction and highlights node and/or formulas that match with the rule (here the formula $(\Diamond q)$ in the world $b2$). You can apply the next rule or finish completely the execution of the tableau method.

## 4 Linking executable code to rule specification

For more expressiveness, in addition of pattern-matching, the user can also use the language Scheme for specifying more precise conditions of the left-hand side

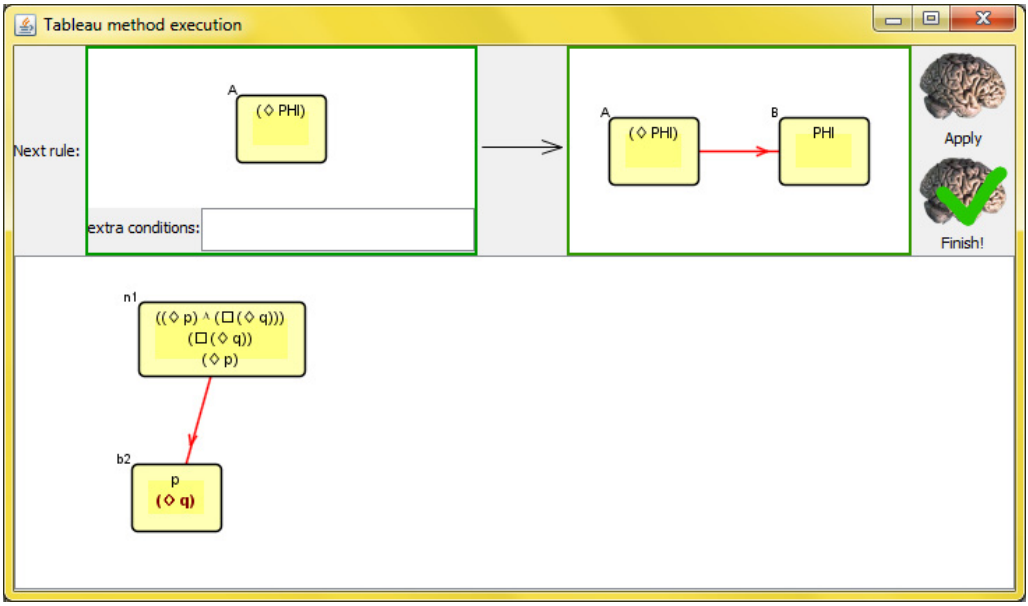



Fig. 2. The window for executing/debugging tableau method

and actions of the right-hand side. LotrecScheme accepts '*all*' [10] expressions of the language Scheme in order to decorate the tableau method rules. Indeed, as the prover is written in Scheme, such expressions are simply evaluated by the function eval and/or the unquote mechanism. More precisely, as LotrecScheme is based on Kawa [3], it accepts 'near-R5RS' Scheme [11] plus extra functions devoted to tableau methods. The software provides help on those extra functions.

We begin with an example dealing with Scheme evaluation in conditions. Then we present examples of tableau methods for modal logics where actions are decorated by executable code thanks to the unquote operations.

### *4.1 Scheme code in conditions. Example of a graded belief logic*

In artificial intelligence, logicians have introduced degrees in modal operators. For instance, in [13], the authors deal with the multi-agent doxastic logic $KD45_n$ extended with the interaction axiom $\Box_j\varphi \rightarrow \Box_i\varphi$ where $i$ and $j$ are integers such that $i < j$. In that logic, the construction $\Box_i\varphi$ stands for "the agent believes at degree $i$ that $\varphi$ is true". In order to implement a tableau method, we need to write the following rule

$$\frac{(degree\ i)(degree\ j)(WR_iV)}{(WR_jV)} \text{where } j > i$$

saying that if $wR_iv$ then $wR_jv$ when $j > i$. In LoTREC this kind of rules is impossible to write since we cannot deal with integers. In LotrecScheme, you write

[10] Malicious logicians incidentally may install trojans and viruses by applying tableau rules!
[11] See http://www.gnu.org/software/kawa/internals/index.html

the following rule:

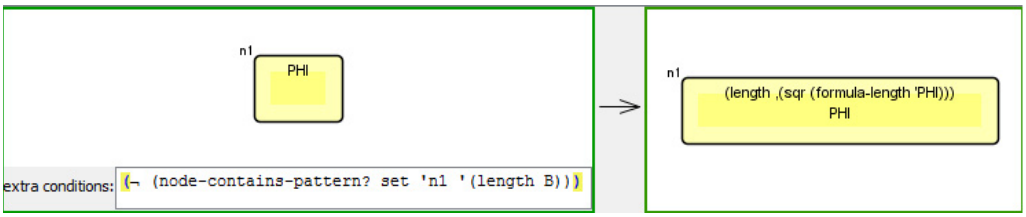
```
(primitiverule
  (condition ((degree I) (degree J) (W link I V)))
  (suchthat (> J I))
  (add (W link J V)))
```

LotrecScheme searches for the pattern (degree I)(degree J) (W link I V) and adds (W link J V) only if $J > I$, where $W, V$ are worlds variables and $I, J$ are integers variables.

### *4.2 Scheme code in conditions and in actions. Example of S4 by limiting the length of a branch*

Logic S4 is the modal logic corresponding to the Kripke frames that are reflexive and transitive. A way for solving the satisfiability problem of S4 may rely on the following fact: a formula $\varphi$ is S4-satisfiable if, and only if it is satisfiable in a transitive and reflexive tree where the number of nodes on a branch is bounded by $|\varphi|^2$, that is the square of the length of $\varphi$ and where some backedges are added between one leaf and one of its ancestor ([12], Lemma 6.13).

In LotrecScheme, it is possible to stop the computation when the depth of nodes is greater than $|\varphi|^2$. The following rule adds the term $(length\ |\varphi|^2)$ in the root of the tableau $(n1)$.




This rule is not implementable in LoTREC for two reasons. First, the rule computes $|\varphi|^2$ and the language of LoTREC is not rich enough to order the computation of $|\varphi|^2$. Secondly, the rule refers to the root node whose name $n1$ and thus there is no pattern-matching with the name of the node. In LoTREC there is always pattern-matching for a node name.

In this rule, the extra-condition states that the node $n1$ does not contain a term of the form $(length\ B)$.

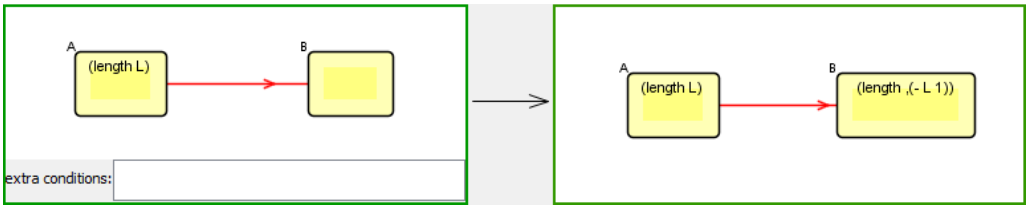
Now, let us consider how we add $(length\ |\varphi|^2)$ in LotrecScheme. We add the following term:

```
(length ,(sqr (formula-length 'PHI)))
```

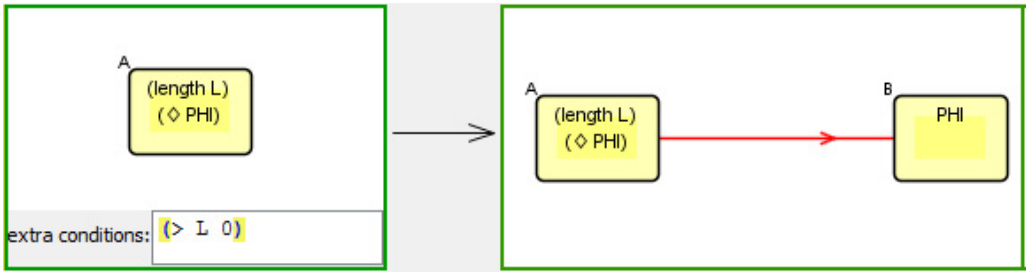
In order to perform evaluations of Scheme expressions in action, we use the unquote operation (the comma) which says that what follows is no longer an abstract term but has to be evaluated. sqr is the function for computing the square of a number. formula-length is a function for computing the number of symbols in a formula. The quote operation (the apostrophe) says to stop to evaluate: indeed, if

$\varphi$ is (diamond p), diamond is not a function!

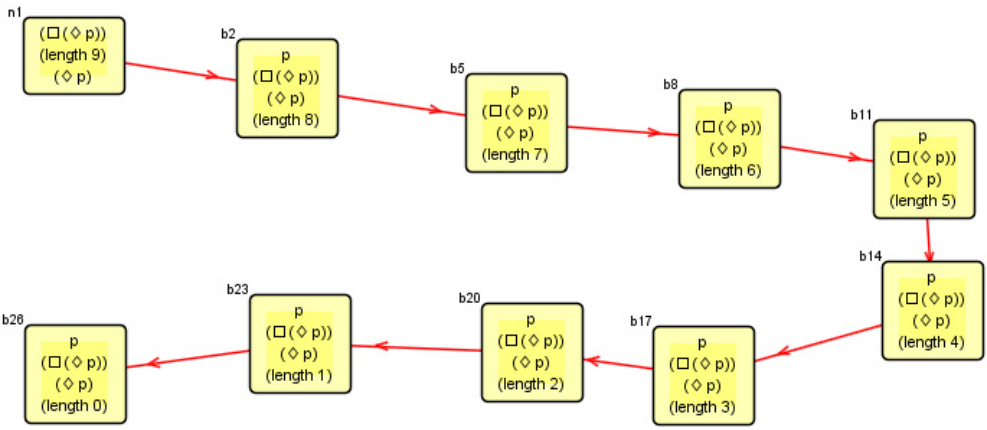
The following rule propagates the information about the depth, that is, if a node contains (length L) then a successor contains (length M) where $M = L - 1$:




Finally, the $\Diamond$-rule only occurs if the integer $L$ in (length L) is strictly positive:




We can test if the formula $\Box\Diamond p$ is S4-satisfiable. We obtain the following tableau:




The starting node of this tableau is $n1$. Other nodes are created by the $\Diamond$-rule until the depth of the node is such that length 0 appears in the node. At this point, the method stops.

### *4.3 Executable code in conditions and in actions. Public announcements*

Public announcement logic [15] is a modal logic with operators $K_i\varphi$ (agent $i$ knows $\varphi$) and $[\psi]\varphi$ (after the public announcements of $\psi$, $\varphi$ holds). The semantics of $K_i\varphi$ is usual and the semantics of $[\psi]\varphi$ is $\mathcal{M}, w \models [\psi]\varphi$ iff $\mathcal{M}, w \models \psi$ implies $\mathcal{M}^\psi, w \models \varphi$ where the updated model $\mathcal{M}^\psi$ is the model $\mathcal{M}$ where we only keep worlds where $\psi$ is true. In the tableau method of Public Announcements Logic [2],

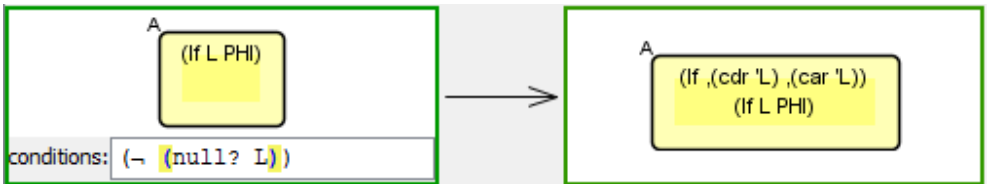



Fig. 3. Rule for announcement.

we need constructions of the form (If L $\varphi$) meaning that the current world survives the successive updates $L$ (a list of formulas) and then $\varphi$ holds in the updated model.

The rule of Figure 3 is a way of implementing the following property:

"node $A$ survives the successive non-empty updates $L = (\psi_1, \dots \psi_n)$ and then $\varphi$ holds" implies that "node $A$ should survive the updates $(\psi_1, \dots \psi_{n-1})$ and then $\psi_n$ holds."

The successive updates $(\psi_1, \dots \psi_{n-1})$ is the tail of $L$ and $\psi_n$ is the head of $L$. The implementation is easily done using usual Scheme primitives: null? to test whether a list is empty, cdr to obtain the tail of a list and car to obtain the head of a list.

## 5 Flexibility of Scheme and merging worlds for S4

The standard way to implement a tableau method for the satisfiability problem of S4 consists in the loop check mechanism. We can detect if there exists two nodes $A$ and $B$ such that $A$ is linked to $B$ and all formulas in $B$ are already included into the node $A$. Usually, provers like LWB, TWB and Lotrec then ban the treatment of formulas in $B$.

In LotrecScheme, we can do more: we can write a tableau rule in order to merge nodes $A$ and $B$ as depicted in Figure 4. Technically, the way of encoding terms of the tableau method and the Scheme language are so flexible that merging nodes only consists in replacement in the whole set of terms $S$, which is also an s-expression $S$ (see Figure 5). The right hand-side shows the action of merging.

The advantage is readability of the resulting model, easing the debugging and the pedagogical demonstration. This fits our objective: to construct a model for a formula and not just a formal structure that can be transformed into a model.

## 6 Conclusion and perspectives

We believe that dynamic functional languages, as Scheme, are suitable for a generic tool for implementing tableau methods for many reasons. The first one is that it offers the compulsory flexibility needed to capture the variety of modal operators and terms that occurred in tableau methods. It enables to offer generic ability, for instance merging two nodes with a simple function. The second one is that such languages often offers an eval function enabling the program to evaluate itself expressions given by the user as an input.

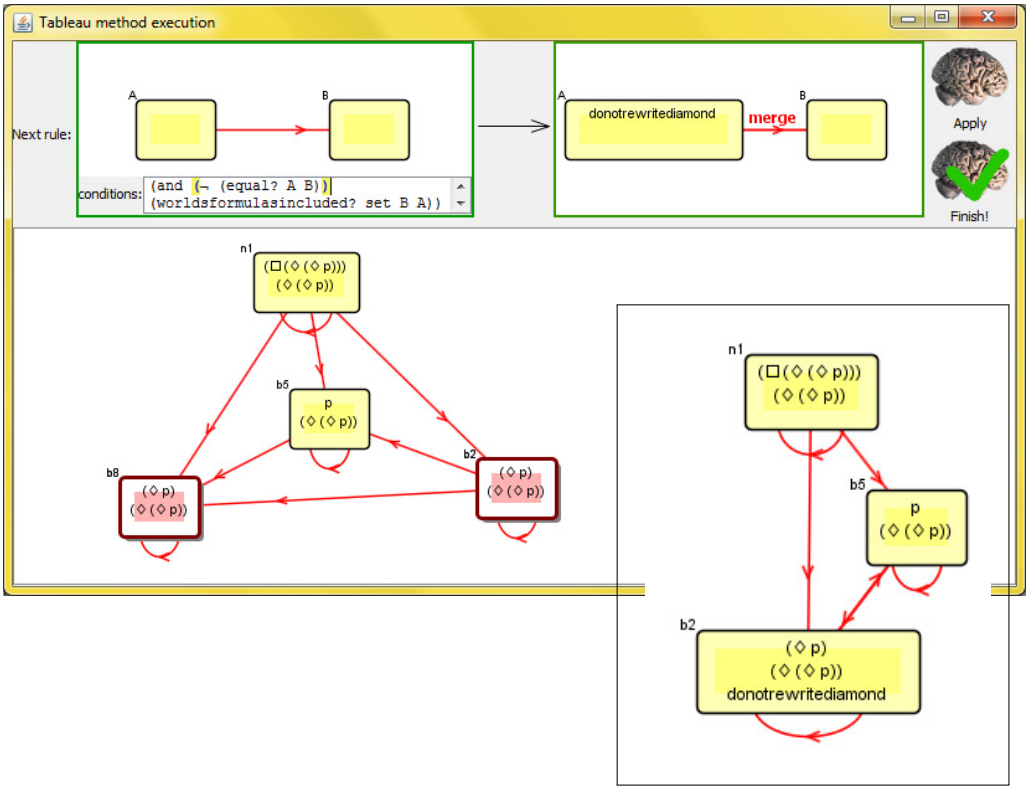



Fig. 4. The tableau method of S4 launched for $\Box\Diamond\Diamond p$. The last step consists in apply the merging rule: node $b2$ and $b8$ will be merged because they fit the condition written in Scheme and are fused in $b2$.

```
(define (merge S e1 e2)
    (if (equal? S e2)
        e1
        (if (list? S)
            (if (null? S)
                '()
                (cons (merge (car S) e1 e2)
                      (merge (cdr S) e1 e2)))
            S)))
```

Fig. 5. The Scheme method merge that replaces all occurence of $e1$ in $S$ by $e2$

A further step is to focus in several logics of agency because the prover is suitable for dealing with coalitions of agents: ATL, STIT-logic, etc.

The graphical user interface is going to be improved in order to be able to edit directly a tableau method with abstract terms like (agent a) etc.

We also want to add functions written in Scheme as a model-checking of first order logic over the finite set of terms that represents the current tableau. It would ease the writing of some conditions.

Another issue is to fill the gap between LoTREC and LotrecScheme concerning the strategy, that is to have more control over the order of the execution of the rules as in LoTREC.

This tool is at its beginning. From now, LotrecScheme uses the same optimization as LoTREC that is to say an event-driven pattern matching [10] but it would be interesting to optimize the pattern-matching by using adapted hash-tables and then to compare the efficiency of LotrecScheme with other existent efficient tableau provers like LWB and TWB.

Another issue is to focus and the correctness of the rewriting system of the tool. The use of an automatic prover like PVS [14] may be investigated.

It would be also interesting to deal with non-terminating tableau methods and transform LoTRECScheme into a prover assistant for tableau methods. The user may decide which pattern to rewrite and which right pattern to apply for a non-deterministic rule.

## References

[1] Abate, P. and R. Gore, *System Description: The Tableaux Work Bench In: Proc. TABLEAUX 2003, LNAI* (2003).

[2] Balbiani, P., H. van Ditmarsch, A. Herzig and T. de Lima, *A tableau method for public announcement logics*, Automated Reasoning with Analytic Tableaux and Related Methods (2007), pp. 43–59.

[3] Bothner, P., *Kawa: compiling dynamic languages to the Java VM*, in: *Proceedings of the annual conference on USENIX Annual Technical Conference*, USENIX Association, 1998, p. 41.

[4] D'Agostino, M., "Handbook of tableau methods," Kluwer Academic Publishers, 1999.

[5] Dybvig, R., "The SCHEME programming language," The MIT Press, 2003.

[6] Fariñas del Cerro, L., D. Fauthoux, O. Gasquet, A. Herzig, D. Longin and F. Massacci, *Lotrec: a generic tableau prover for modal and description logics (regular paper)*, in: R. Goré, A. Leitsch and T. Nipkow, editors, *International Joint Conference on Automated Reasoning (IJCAR'01) , Siena, Italy, 18/06/2001-23/06/2001*, LNCS **2083** (2001), pp. 453–458, presentation of the modal generic prover LoTREC.
URL http://www.irit.fr/~Andreas.Herzig/P/ijcar01.html

[7] Fariñas Del Cerro, L., O. Gasquet, A. Herzig and M. Saade, *LoTREC: An environment for experiencing Kripke Semantics*, in: M. Manzano, B. P. Lancho and A. Gil, editors, *International Congress on Tools for Teaching Logic (ICTTL), Salamanca, Spain, 26/09/06-30/09/06* (2006), pp. 41–44, iSBN: 84-690-0348-8.

[8] Gasquet, O., A. Herzig, D. Longin and M. Saade, *LoTREC: Logical Tableaux Research Engineering Companion*, in: B. Beckert, editor, *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005), Koblenz, Germany, 14/09/05-17/09/05* (2005), pp. 318–322.
URL http://www.irit.fr/recherches/LILAC/Lotrec

[9] Gasquet, O., A. Herzig and M. Saade, *Terminating modal tableaux with simple completenness proof*, , **6** (2006), pp. 167–186, iSBN 1904987222.
URL http://www.aiml.net/volumes/volume6/Gasquet-Herzig-Sahade.ps

[10] Gasquet, O., B. Said and F. Schwarzentruber, *A semantics for an event based generic tableau prover (student paper)*, in: *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX), Oslo, Norway, 06/07/2009-10/07/2009* (2009), pp. 1–15.
URL ftp://ftp.irit.fr/IRIT/LILAC/Reports/Tableaux2009RT.pdf

[11] Heuerding, A., G. Jäger, S. Schwendimann and M. Seyfried, *The logics workbench LWB: A snapshot*, Euromath Bulletin **2** (1996), pp. 177–186.

[12] Joseph Y. Halpern, Y. M., *A guide to completeness and complexity for modal logics of knowledge and belief* (1996).

[13] Laverny, N. and J. Lang, *From Knowledge-Based Programs to Graded Belief-Based Programs, Part I: On-Line Reasoning*, Uncertainty, Rationality, and Agency (2006), pp. 223–267.

[14] Owre, S., J. Rushby and N. Shankar, *Pvs: A prototype verification system*, Automated Deduction—CADE-11 (1992), pp. 748–752.

[15] Plaza, J., *Logics of public communications*, in: *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems: Poster Session Program* (1989), pp. 201–216.

[16] Said, B., "Réécriture de graphes pour la construction de modèles en logique modale," Thèse de doctorat, Université de Toulouse, Toulouse, France (2010).
URL
ftp://ftp.irit.fr/IRIT/LILAC/Theses_et_habilitations/2010_Bilal_SAID_thesis-en-fr.pdf

[17] van Valkenhoef, G., E. van der Vaart and R. Verbrugge, *OOPS: An S5n Prover for Educational Settings*, Electronic Notes in Theoretical Computer Science **262** (2010), pp. 249–261.